Available online at www.sciencedirect.com
ScienceDirect
Procedia Computer Science 1 (2012) 1531–1534
Procedia Computer Science
www.elsevier.com/locate/procedia

International Conference on Computational Science, ICCS 2010

Position paper on the simulation of high-frequency optical waves

K. Hertel[a,b,*], C. Pflaum[a,b]

[a] *Department of Computer Science 10 (System Simulation), University of Erlangen-Nuremberg, Cauerstraße 6, 91058 Erlangen, Germany*
[b] *Erlangen Graduate School in Advanced Optical Technologies (SAOT), Paul-Gordan-Str. 6, 91052 Erlangen*

Abstract

This paper will provide an insight into the design and software engineering aspects of a simulation software for time-harmonic electro-magnetic waves with application to the simulation of optical waves in lithography and thin-film solar cells. This design is oriented towards applicability to large systems, as it is driven by the computationally challenging need to simulate large structures in order to tackle real-world problems. We will present an overview of the organizational aspects and the interaction of the components involved.

Keywords: Simulation, Optical Waves, Maxwell, FDFD, Expression Templates

1. Some Background

Solving Maxwell's equations numerically requires at least 20 nodes per wavelength on the discretized mesh in order to be able to obtain usable results. In the case of solar cells, we're dealing with visible light for the most part, resulting in relatively short wavelengths of roughly 300 - 1100nm. On the other hand, dimensions of a solar cell are orders of magnitude larger, which results in huge computational domains. The discretization scheme used in this software is based on the finite difference frequency domain method, thus resulting in stencil operations which do not require any system matrices to be stored, but only vector components of the electric and magnetic fields for every node of the discretization. For example, Ampère's law

$$\epsilon\, \partial_t E = \nabla \times H - \sigma E \tag{1}$$

in its finite difference frequency domain notation can be restated as follows:

$$\left(1 + \rho\frac{\tau\sigma}{\epsilon}\right) E^{(n+1)\tau} e^{i\omega\tau} = E^{n\tau} + \frac{\tau}{\epsilon}\left(\nabla_h \times H^{(n+\frac{1}{2})\tau}\right) e^{i\omega\frac{1}{2}\tau} - \frac{\tau\sigma}{\epsilon}(1-\rho)\, E^{n\tau} + \tau S_E, \tag{2}$$

where $\nabla_h\times$ represents the finite difference curl operator on a staggered grid (see [1]). Since the number of nodes used in simulations typically ranges in the 10s of millions, both computational and memory requirements are enormous. Therefore parallelization, and with it the use of distributed memory, is a key aspect of the implementation.

*Corresponding author
Email addresses: kai.hertel@informatik.uni-erlangen.de (K. Hertel), pflaum@informatik.uni-erlangen.de (C. Pflaum)

1877-0509
doi:10.1016/j.procs.2010.04.171

2. Software Design

Modularization and the use of several layers of abstraction seem appropriate, as the iterative solver logic is largely independent of the application. For example the specific equations to be solved on the one hand, and the underlying geometry of the computational domain on the other hand are not directly related to one another. Therefore, a generic library provides the structures necessary to describe the computational domain in an orthogonal finite difference mesh and to represent both the variables on the domain and the operators acting thereon. The key technique used for implementing operators on mesh variables (i.e. vectors) is expression templates. A major benefit is that they allow for the equations and iterator logic to use natural and simple expressions and hide most of the underlying parallelization scheme from the application layer.

For parallelization MPI is being used, which is embedded into the library operations directly for the most part. During initialization of the mesh, partitions are assigned to available compute nodes by the application layer. As the discretization of solid objects like solar cells is static and adaptivity is not necessarily required, balancing the workload is a relatively straight-forward task. Synchronization on the partition boundaries on the other hand is necessary after every step in the iteration due to the stencil operations involved. This is implemented in the expression template library, so the application layers do not have to be concerned with it.

The main function of the application layers is to act as an interface to the user and to allow for the domain to be parametrized in such a way, that regions of different materials can be set up. Their electromagnetic properties can then simply be stored in the form of a vector of coefficients on the mesh in much the same way as the primary variables, the vectors for the electric and magnetic field components, are stored. The description of the setting can generally be more complex than just defining a stack of layers of different materials. This is especially true where the geometry and physical structure of their interfaces is important. Therefore, this part of the code is represented by a separate module that interfaces with classes describing and representing the specific problem settings accurately.

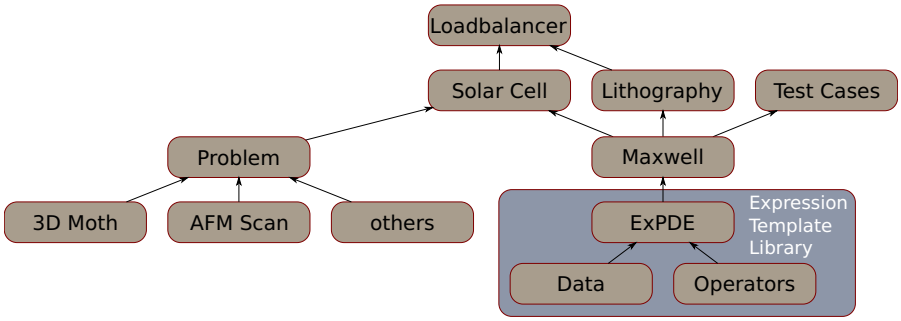

Figure 1: Structure Diagram of the Module Layers and their Dependencies

The iterative evolution of the EM field is implemented in a rather compact manner. To this end, the iterative Maxwell solver uses expression template operator logic.

```
Ey = Ct_Ey * Ey + Cs_Ey *
     (T(Hx) - B(Hx) + W(Hz) - E(Hz)) + S_Ey;
```

The snippet above describes part of the iteration scheme for one component of the electric field and can be interpreted quite intuitively when compared to the finite difference notation (2) above.

3. Testing

For the basic layers, the use of debuggers turns out to be an adequate instrument. Thus, unit and integration testing of available modules is currently implemented only on the more complex and thus error-prone layers. The library

for example is naturally designed to be tested against various sample applications. This way it is easily ensured that both data structures and operator logic stay consistent during development. The Maxwell solver can be tested against settings that consist of a problem statement for which an exact analytical solution can be derived. This analytical solution is then used as a basis for comparison to the numerical solution of the solver. In this way we measure the convergence of the iterative method towards the exact solution. This is mostly useful when the simulation code is extended for new scenarios and physical setups.

4. Expression Templates

Using a library based on expression templates allows for a quick and convenient implementation of the application logic, as grid iteration operators and the basic tools for differential operators are all provided by the library layers (see [2], [3]). The resulting code is relatively fast, since most of the operations can be inlined very efficiently by the compiler's preprocessor. This in turn yields good optimization characteristics at compile time and results in high application performance (see [4]). Nevertheless, when compared to manually optimized C++ codes, there may be a noticable drop in performance with the classical expression template based approach, which we have not investigated conclusively yet for this application. However, due to the ease of transition we are considering a shift to fast expression templates which should result in the performance drop to become negligible compared to non-expression-template based implementations (cf. [2]). The use of expression templates, despite its character as a convenient tool for modeling application logic on a very high level, also introduces some drawbacks to the implementation with regards to maintainability though. For once, debugging and profiling programs that make extensive use of template classes is a challenge. That problem is somewhat mitigated, or depending on the vantage point of the user maybe enhanced, by the fact that debugging parallel applications is not an easy task to begin with. Therefore, tool support is of the essence when it comes to these kinds of tasks. Luckily spotting the parts of the code responsible for performance bottlenecks is relatively simple in general for the kind of application presented. This is because the implementation of the evolution of the equations described is easily identified and traced through the underlying layers. From a designer's point of view the principles employed by template-based programming and the use of expression templates yields a great deal of benefits to the software engineering process.

5. Current Progress

In order to study the effects of expression templates on optimization in this application setting, we have begun implementing the stencil operations described above in a small C-based kernel that will work on the same data as the template-based version. First results seem to indicate, that optimization of decoupled C code may hold slight advantages in runtime, at least for serial runs, as they do not strain the capabilities of available compiler technology to the same extent. In a parallel setting, these effects are harder to pinpoint though, as MPI communication and the latencies and synchronization issues related to it appear to play a rather significant part. A more detailed analysis will need to be conducted in order to support these preliminary findings more conclusively. One way to resolve possible performance drawbacks introduced by expression-template based codes may be a shift to fast expression templates as described in [2].

6. Conclusion

As we have seen in this work, the use of expression templates can greatly aid the design of scientific simulation software as it helps in modeling physical and mathematical contexts in an intuitive way as well as encapsulating the underlying processes in an easy-to-use, self-contained, abstract interface. Using this technique, applications can easily implement numerical methods even in large-scale parallel settings without losing control over the principle of separation of concerns.

7. Acknowledgements

The authors gratefully acknowledge funding from the Erlangen Graduate School in Advanced Optical Technologies (SAOT) by the German National Science Foundation (DFG) in the framework of the German Excellence Initiative and the Bavarian Competence Network for Technical and Scientific High Performance Computing (KONWIHR).

8. References

[1] A. Taflove, S. Hagness, Computational Electrodynamics — The Finite-Difference Time-Domain Method, Artech House, Boston, London, 2000.
[2] J. Härdtlein, C. Pflaum, A. Linke, C. H. Wolters, Advanced expression templates programming, Computing and Visualization in Science 12 (2009) 59–68. doi:10.1007/s00791-009-0128-2.
[3] C.Pflaum, Z. Rahimi, Automatic parallelization of staggered grid codes with expression templates, International Journal on Computational Science and Engineering 4 (4) (2009) 306–313. doi:10.1504/IJCSE.2009.029166.
[4] C. Jandl, K. Hertel, W. Dewald, C. Pflaum, High performance computing for the simulation of thin-film solar cells, High Performance Computing in Science and Engineering, Garching 2009(to appear).